

A Bayesian Approach to Predicting NFL Quarterback Scores in Fanduel Tournaments

STAT 578, Fall 2017, Team 5: Aaron Ray, Kiomars Nassiri, Michael Chan

October 25, 2017

Project Description

The National Football League (NFL), being one of the major professional sports leagues in North America, has a wide audience. participates in the NFL craze by competing in fantasy football tournaments organized by the daily fantasy site, “FanDuel.com”. Participants in a **Fantasy Football** game act as the managers of a virtual football team and try to maximize their points by picking up the best line-up. Points are given based on actual performance of players in real-world competition. For the purpose of this project we have chosen to work with the data gathered from the **FanDuel** internet company. We will leverage a Hierarchical Bayesian approach with the Markov Chain Monte Carlo method to predict the fantasy points likely to be scored by an NFL quarterback in any given game. The goal is to predict the points scored by each player given certain prior conditions and predictor variables that will assist our model in providing credible posterior prediction intervals.

The analysis is inspired by the study presented in the article, **Bayesian Hierarchical Modeling Applied to Fantasy Football Projections for Increased Insight and Confidence**, by Scott Rome.

Team Members

- **Aaron Ray** (aaronwr2@illinois.edu)*
- **Kiomars Nassiri** (nassiri2@illinois.edu)
- **Michael Chan** (mhchan3@illinois.edu)

*Contact Person

Dataset Description

Team has set up a process to gather the historical data from the RotoGuru website. The following is the code used to get the data from RotoGuru:

```
# Scrape rotoguru1 site for weekly FanDuel stats and bind each week's data to the
# pre-defined dataframe, 'd'.

for(year in 2014:2017){
  for(week in 1:16){
    page = read_html(
      gsub(" ", "", ,
           paste("http://rotoguru1.com/cgi-bin/fyday.pl?week=", week, "&year=",
                 year, "&game=fd&scsv=1"))
    )
    dtext = page %>% html_nodes("pre") %>% html_text(trim = TRUE)
    dtable = read.table(text=dtext, sep = ";", header=TRUE, col.names = cnames,
```

```

    quote=NULL)
d = rbind(d,dtable)
}
}

```

Data cleaning is performed using R routines. Some data cleaning tasks are needed to calculate Player rank.

Response Variables

- **FanDuelPts**: Points position at the end of a single game

Predictor Variables

- **AvgPts5Wks**: The 5 game average points of the player
- **AvgOppPAP7Wks** : The 7 game average Opposing Points Allowed to Position (OppPAP) by the current player's opposing defense. For example, if the Buffalo Bills defense allowed a total of 30 points per game to wide receivers for six games straight, then this number would equal to the average of 30 for any wide receiver facing the Bills defense.
- **Position**: The position the player plays
- **HomeGame**: Whether it is home game.
- **Rank**: The rank of a player based on recent performance

Analysis Ideas

Model

At the lowest level, we model the performance (**FanDuelPts**) as normally-distributed around a true value:

$$y|\alpha, \beta_{defense}, \beta_{home}, \beta_{away}, \sigma_r^2 \sim N(\alpha + X_{defense} \cdot \beta_{defense} + X_{home} \cdot \beta_{home} + X_{away} \cdot \beta_{away}, \sigma_y^2 I)$$

where

α = The average fan duel point of the previous 5 weeks of the player, **AvgPts5Wks**

$\beta_{defense,p}$ = defense coefficient against team t for position p

$\beta_{home,p,r}$ = home coefficient for position p and a rank r player

$\beta_{away,p,r}$ = Away coefficient for position p and a rank r player

y = **FanDuelPts**

x_p = interaction indicator term for opposing team score allowed by position p

$x_{home,p,r}$ = interaction indicator term for rank r, position p, and whether it is home game

At higher level, we model the defense effect, $\beta_{defense}$, as how good(bad) a particular team's defense is against the player's position. We pool the effect based on the position of the player. That is, the defense coefficient is normally distributed from the same position specific distribution.

$$\beta_{defense,p} \sim N(\delta_p, \sigma_\delta^2)$$

where σ_δ is constant = 1000

For the home and away game effect, β_{home} and β_{away} , we model the effect for player of the same rank has the same distribution. We model the home and away game effect to be the same for players of the same position.

$$\beta_{home,p,r} \sim N(\eta_r, \sigma_\eta^2)$$

$$\beta_{away,p,r} \sim N(\rho_r, \sigma_\rho^2)$$

where σ_η, σ_ρ are constant = 1000

We will approximate non informative prior using:

$$\sigma_y \sim Inv - gamma(0.0001, 0.0001)$$

$$\delta \sim N(0, 10000^2)$$

$$\eta \sim N(0, 10000^2)$$

$$\rho \sim N(0, 10000^2)$$

Here is the JAGS model:

```
#sink("fdp.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                  + inprod(X.home[i, ], beta.home)
                  + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    beta.defense[p] ~ dnorm(delta[p], 1/1000^2)
    delta[p] ~ dnorm(0, 1/10000^2)
  }

  # The entry of the beta.home and beta.away corresponds to Rank:Position
  # In our model, we pool the beta.home/away based on rank
  for (r in 1:Num.Rank) {
    for (t in 1:Num.Position) {
      beta.home[(r-1) * Num.Position + t] ~ dnorm(eta[r], 1/1000^2)
      beta.away[(r-1) * Num.Position + t] ~ dnorm(rho[r], 1/1000^2)
    }
    eta[r] ~ dnorm(0, 1/10000^2)
    rho[r] ~ dnorm(0, 1/10000^2)
  }

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}

#      ",fill = TRUE)
#sink()
```

Sample Data

```

fdp <- read.csv("fdpfinal.csv", sep = ',', header = TRUE)

head(fdp)

##   Position Year YearWeek Opponent Week PlayerId          Name      Team
## 1       QB 2015     201513  Steelers  13    1060 Hasselbeck, Matt  Colts
## 2       QB 2015     201514  Jaguars  14    1060 Hasselbeck, Matt  Colts
## 3       QB 2015     201515  Texans   15    1060 Hasselbeck, Matt  Colts
## 4       QB 2015     201516 Dolphins  16    1060 Hasselbeck, Matt  Colts
## 5       QB 2015     201501 Ravens   1    1081 Manning, Peyton Broncos
## 6       QB 2015     201502 Chiefs   2    1081 Manning, Peyton Broncos
##   HomeGame FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP
## 1         0      6.86        6500      20.95      8.045    17.44
## 2         0      9.08        6600      24.16      6.738    17.44
## 3         1      8.98        6400      16.36      9.690    17.44
## 4         0      3.96        6000      20.46      6.002    17.44
## 5         1      5.90        9100      18.99     11.697    17.44
## 6         0     21.24        8200      13.85      4.342    17.44
##   OallStddevPAP AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks
## 1      2.909      14.85      4.824    Rank1      Rank2
## 2      2.909      14.82      4.897    Rank1      Rank2
## 3      2.909      13.56      5.490    Rank1      Rank3
## 4      2.909      12.11      5.566    Rank1      Rank2
## 5      2.909      14.96      8.496    Rank1      Rank2
## 6      2.909      10.53      5.011    Rank2      Rank3

```

```

fdp['Rank'] = fdp$OffRnk5Wks
fdp['Locality'] = 'Away'
fdp[fdp$HomeGame == 1, 'Locality'] = 'Home'

```

Simple Ideas

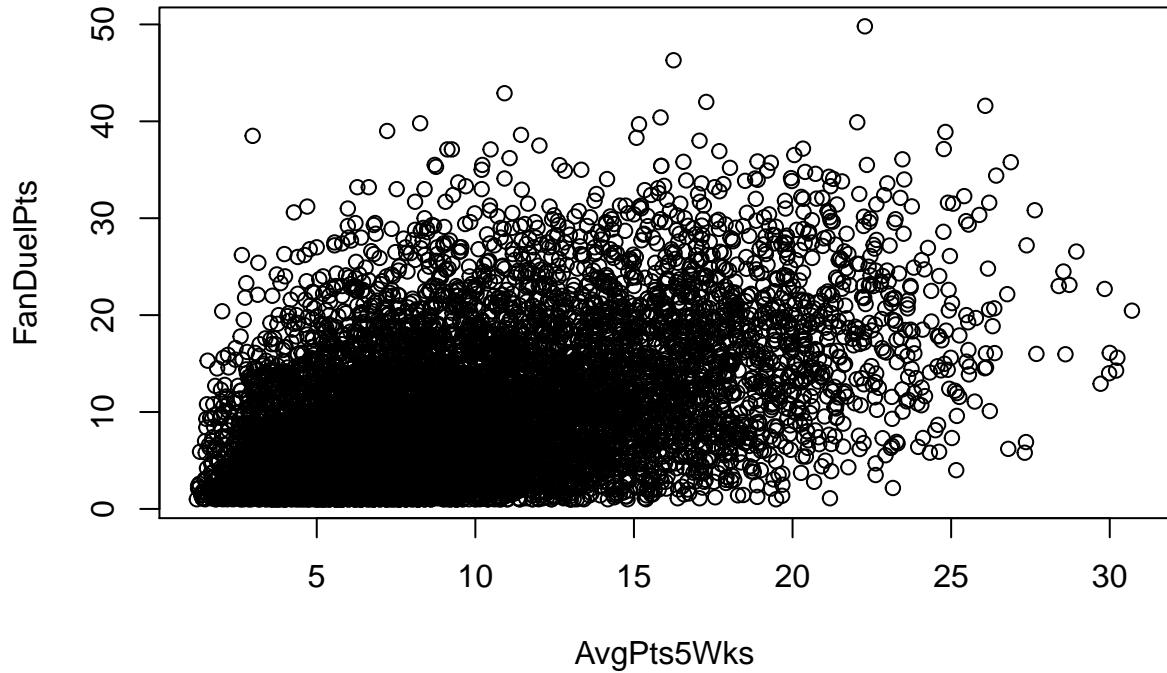
$$y|\alpha \sim N(\alpha, \sigma_y^2 I)$$

```

mod.classic = lm(FanDuelPts ~ AvgPts5Wks, data = fdp)

plot(FanDuelPts ~ AvgPts5Wks, data = fdp)

```



All over the place, let's add the team defense

`X.defense` is the indicator matrix

`** Set up train data **`

```

fdp_train=fdp[fdp$Year == 2015 & (fdp$Position == "QB" | fdp$Position == "PK"), ]
fdp_train = droplevels(fdp_train)

Num.Opponent = length(unique(fdp_train[, "Opponent"]))
Num.Position = length(unique(fdp_train[, "Position"]))
Num.Rank = length(unique(fdp_train[, "Rank"]))

if (Num.Position == 1) {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_train)
  X.home = model.matrix(~ 0 + Rank , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank , data=fdp_train)
} else {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks:Position, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_train)
  X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)
}

X.home = X.home * fdp_train$HomeGame

```

```

X.away = X.away * (1- fdp_train$HomeGame)
X = cbind(X.defense, X.home, X.away)

library(rjags)
set.seed(20171008)

# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv= 0.01, delta = rep(-10000, Num.Position),
        eta = c(10000, -10000, 10000, -10000)[1:Num.Rank],
        rho = c(-10000, 10000, -10000, 10000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv= 0.01, delta = rep(10000, Num.Position),
        eta = c(10000, -10000, -10000, 10000)[1:Num.Rank],
        rho = c(-10000, 10000, 10000, -10000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001, delta = rep(-10000, Num.Position),
        eta = c(-10000, 10000, -10000, 10000)[1:Num.Rank],
        rho = c(10000, -10000, 10000, -10000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001, delta = rep(10000, Num.Position),
        eta = c(-10000, 10000, 10000, -10000)[1:Num.Rank],
        rho = c(10000, -10000, -10000, 10000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)
data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$AvgPts5Wks,
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.Position=Num.Position,
  #Num.Opponent=Num.Opponent,
  Num.Rank=Num.Rank
)
burnAndSample = function(m, N.burnin, N.iter, show.plot, mon.col) {
  update(m, N.burnin) # burn-in

  x <- coda.samples(m, mon.col, n.iter=N.iter)

  if(show.plot) {
    plot(x, smooth=FALSE)
  }

  gelman.R = gelman.diag(x, autoburnin=FALSE, multivariate = FALSE)
  print(gelman.R)

  result <- list(
    coda.sam = x,
    gelman.R.max=max(gelman.R$psrf[, 1])
  )
}

```

```

    return(result)
}

runModel=TRUE
runSample=TRUE

mon.col <- c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq")

NSim = 50000
if (runModel) {
  m <- jags.model("fdp.bug", data.jags, inits = jags.inits, n.chains=4, n.adapt = 1000)
  save(file="fdp.jags.model.Rdata", list="m")
} else {
  load("fdp.jags.model.Rdata")
  m$recompile()
}

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 866
##   Unobserved stochastic nodes: 29
##   Total graph size: 21274
##
## Initializing model

load.module("dic")

## module dic loaded

N.Retry.Loop = 1
if (runSample) {
  N.burnin=2500/2
  for (loopIdx in 1:N.Retry.Loop) {
    (start_time <- Sys.time())
    (N.burnin = N.burnin * 2)
    result = burnAndSample(m, N.burnin, NSim, show.plot=FALSE, mon.col = mon.col)
    (end_time <- Sys.time())
    (result$gelman.R.max)
  }
  save(file=paste("fdp.jags.samples.", N.burnin, ".Rdata", sep=""), list="result")
  save(file=paste("fdp.jags.model.", N.burnin, ".Rdata", sep=""), list="m")
} else {
  N.burnin=2500/2 * (2**N.Retry.Loop)
  load(paste("fdp.jags.samples.", N.burnin, ".Rdata", sep=""))
  load(paste("fdp.jags.model.", N.burnin, ".Rdata", sep=""))

  gelman.diag(result$coda.sam, autoburnin=FALSE, multivariate = FALSE)
}

## Potential scale reduction factors:

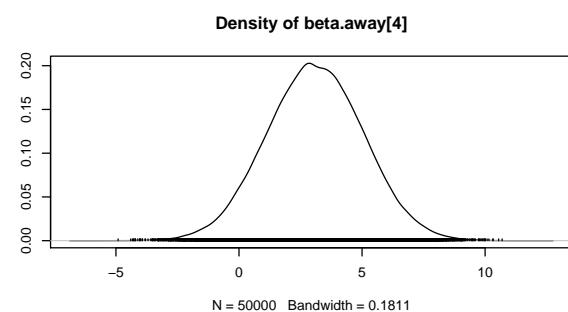
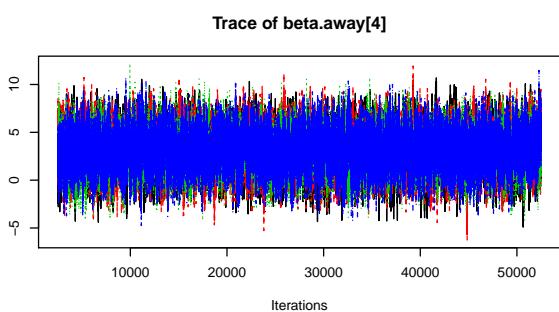
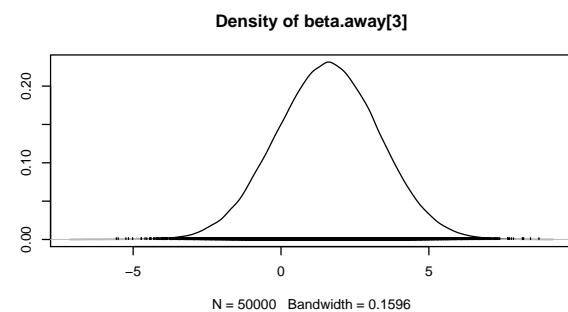
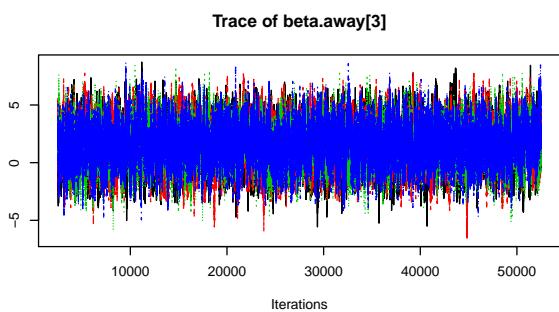
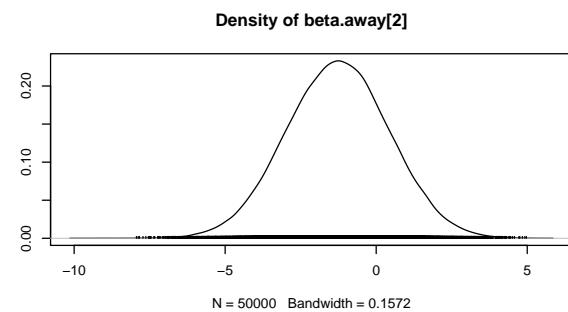
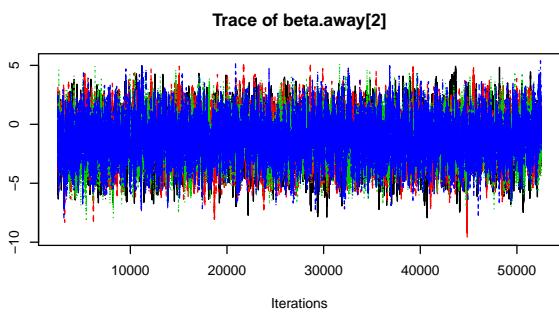
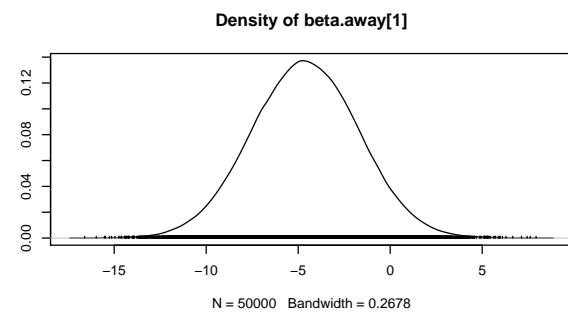
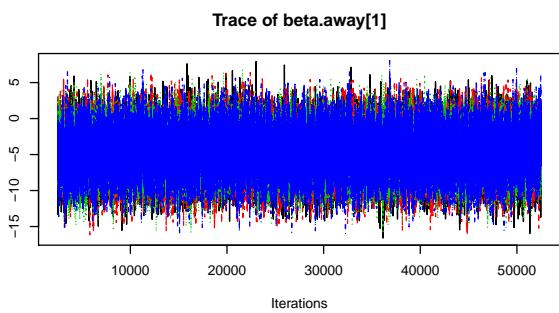
```

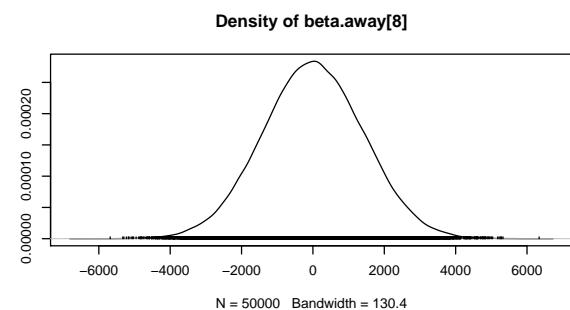
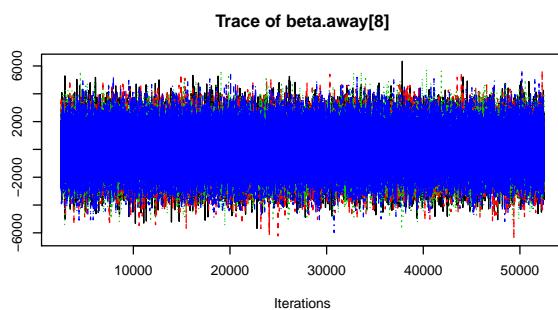
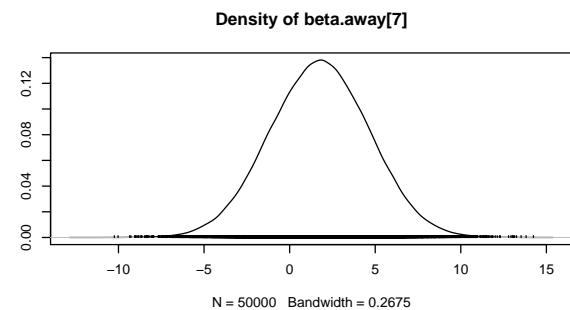
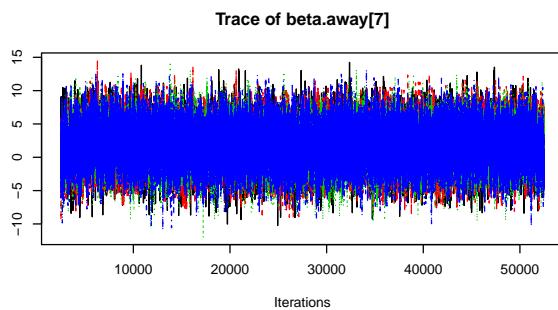
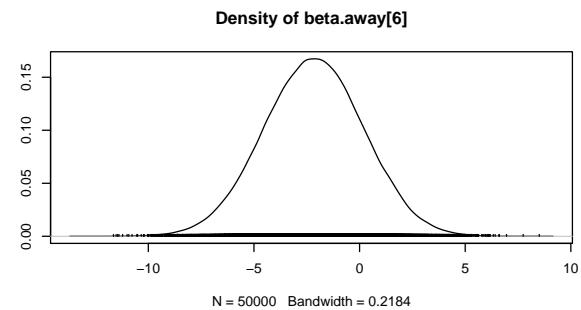
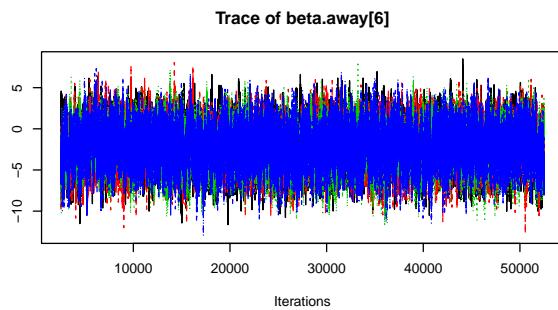
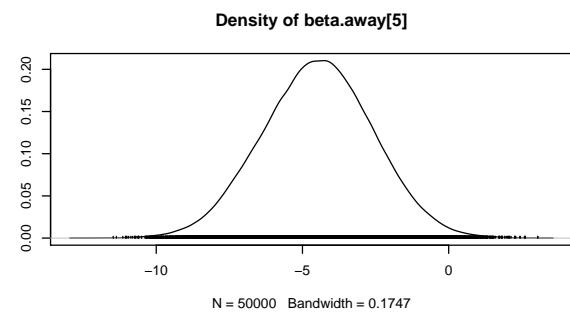
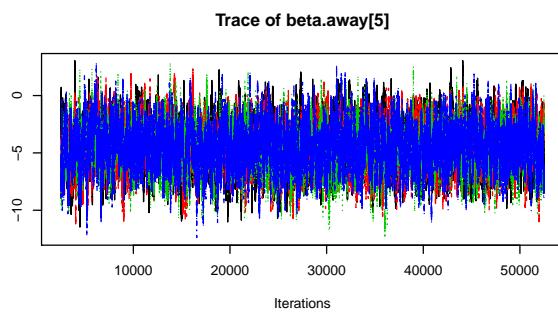
```

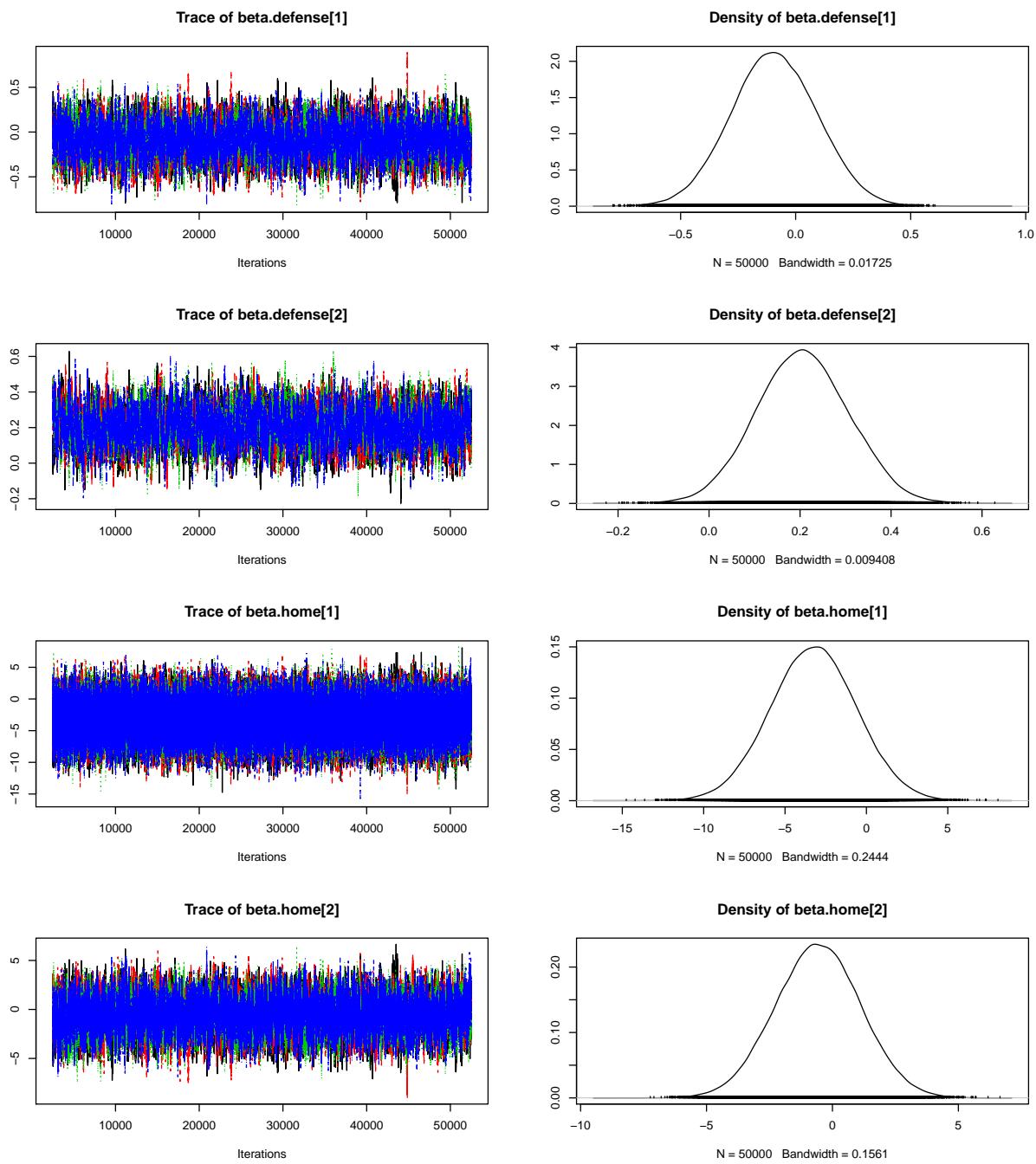
##                                     Point est. Upper C.I.
## beta.away[1]                  1          1
## beta.away[2]                  1          1
## beta.away[3]                  1          1
## beta.away[4]                  1          1
## beta.away[5]                  1          1
## beta.away[6]                  1          1
## beta.away[7]                  1          1
## beta.away[8]                  1          1
## beta.defense[1]                1          1
## beta.defense[2]                1          1
## beta.home[1]                  1          1
## beta.home[2]                  1          1
## beta.home[3]                  1          1
## beta.home[4]                  1          1
## beta.home[5]                  1          1
## beta.home[6]                  1          1
## beta.home[7]                  1          1
## beta.home[8]                  1          1
## delta[1]                      1          1
## delta[2]                      1          1
## eta[1]                        1          1
## eta[2]                        1          1
## eta[3]                        1          1
## eta[4]                        1          1
## rho[1]                        1          1
## rho[2]                        1          1
## rho[3]                        1          1
## rho[4]                        1          1
## sigmasq                       1          1

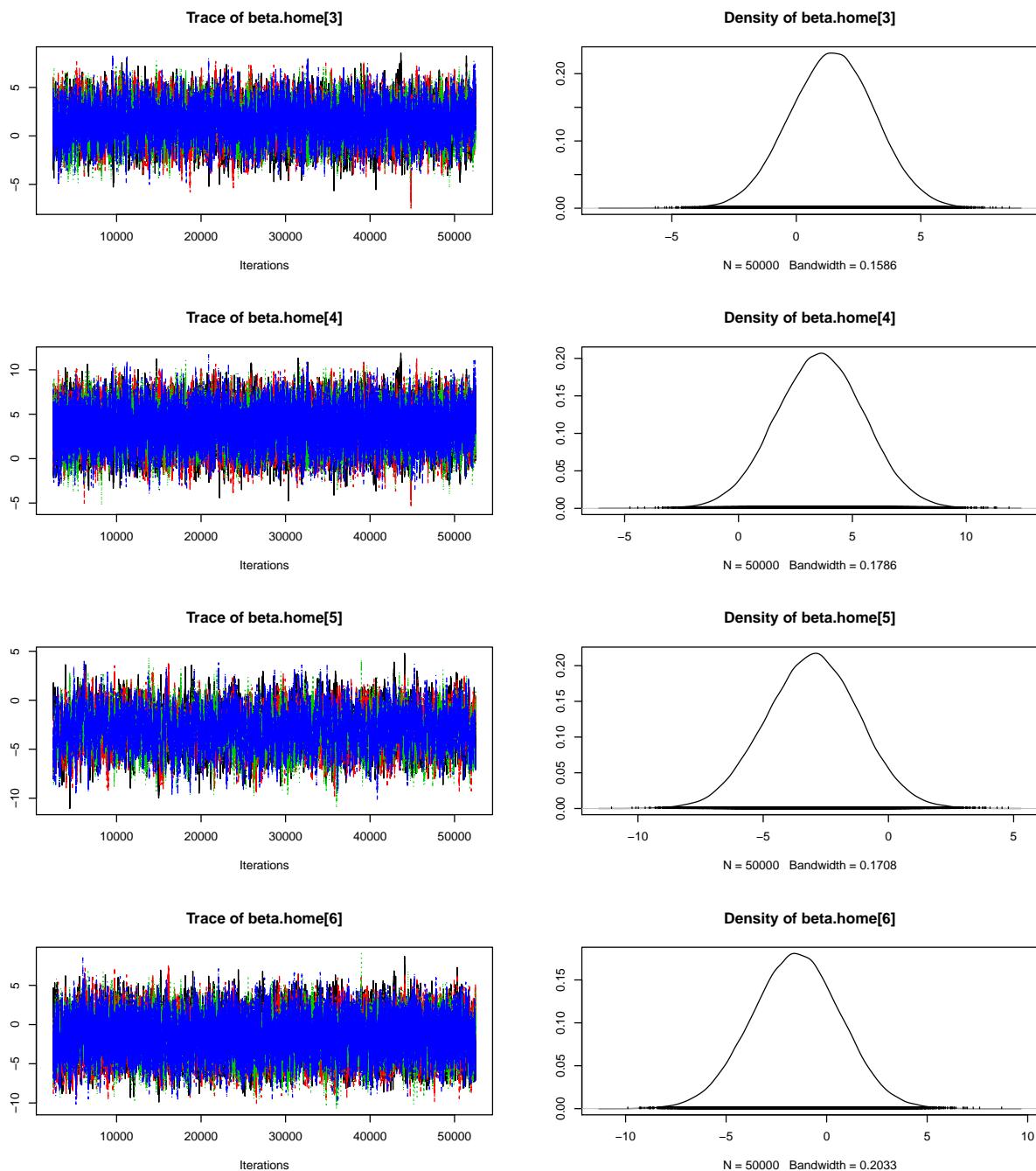
```

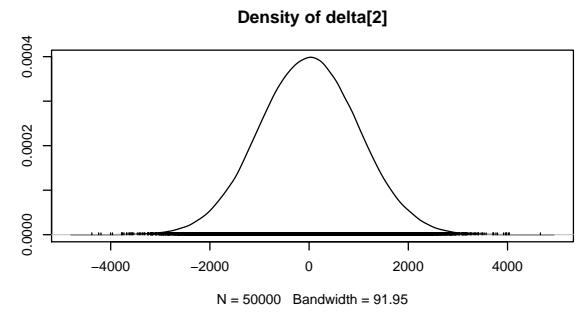
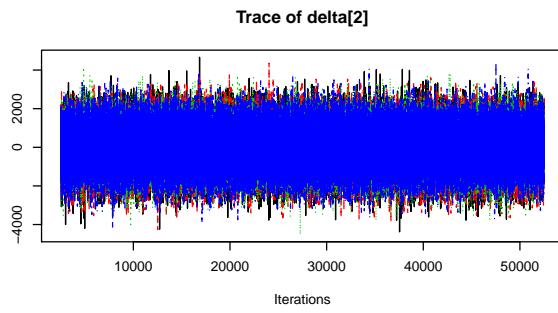
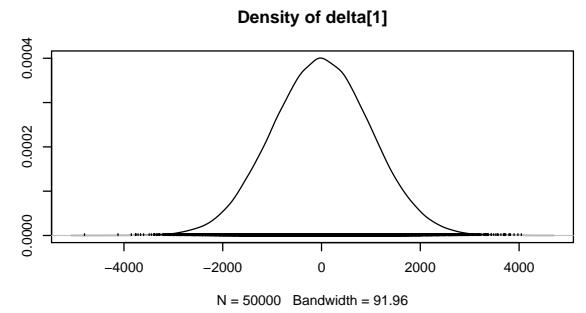
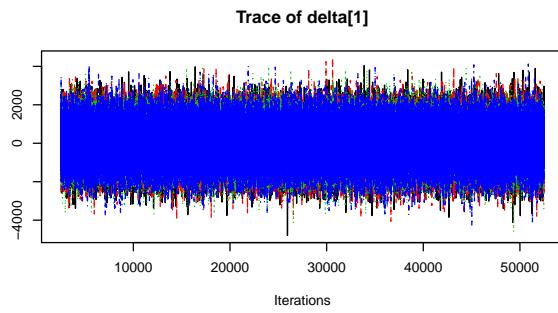
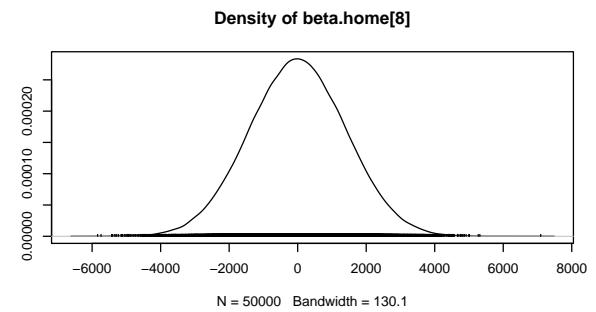
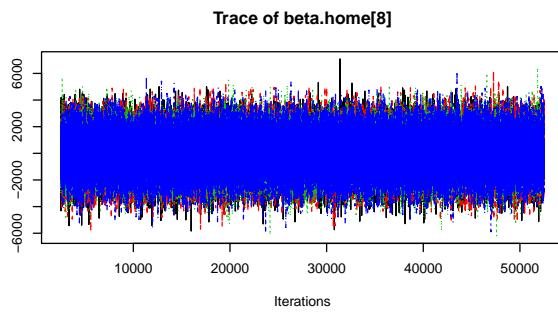
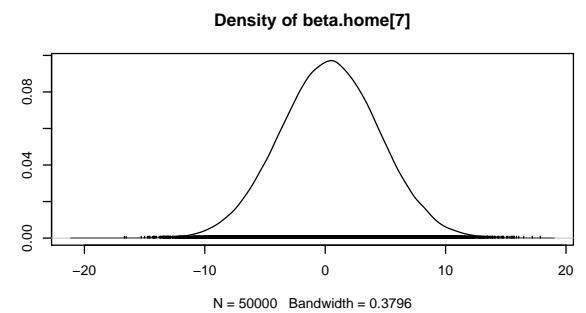
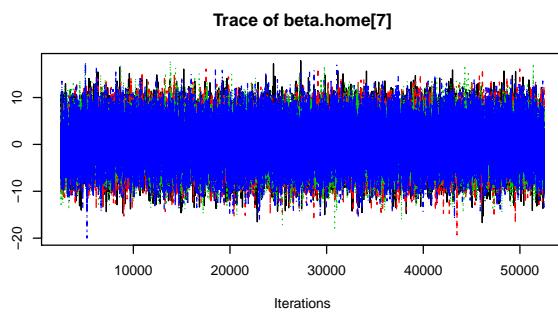
```
plot(result$coda.sam, smooth=FALSE)
```

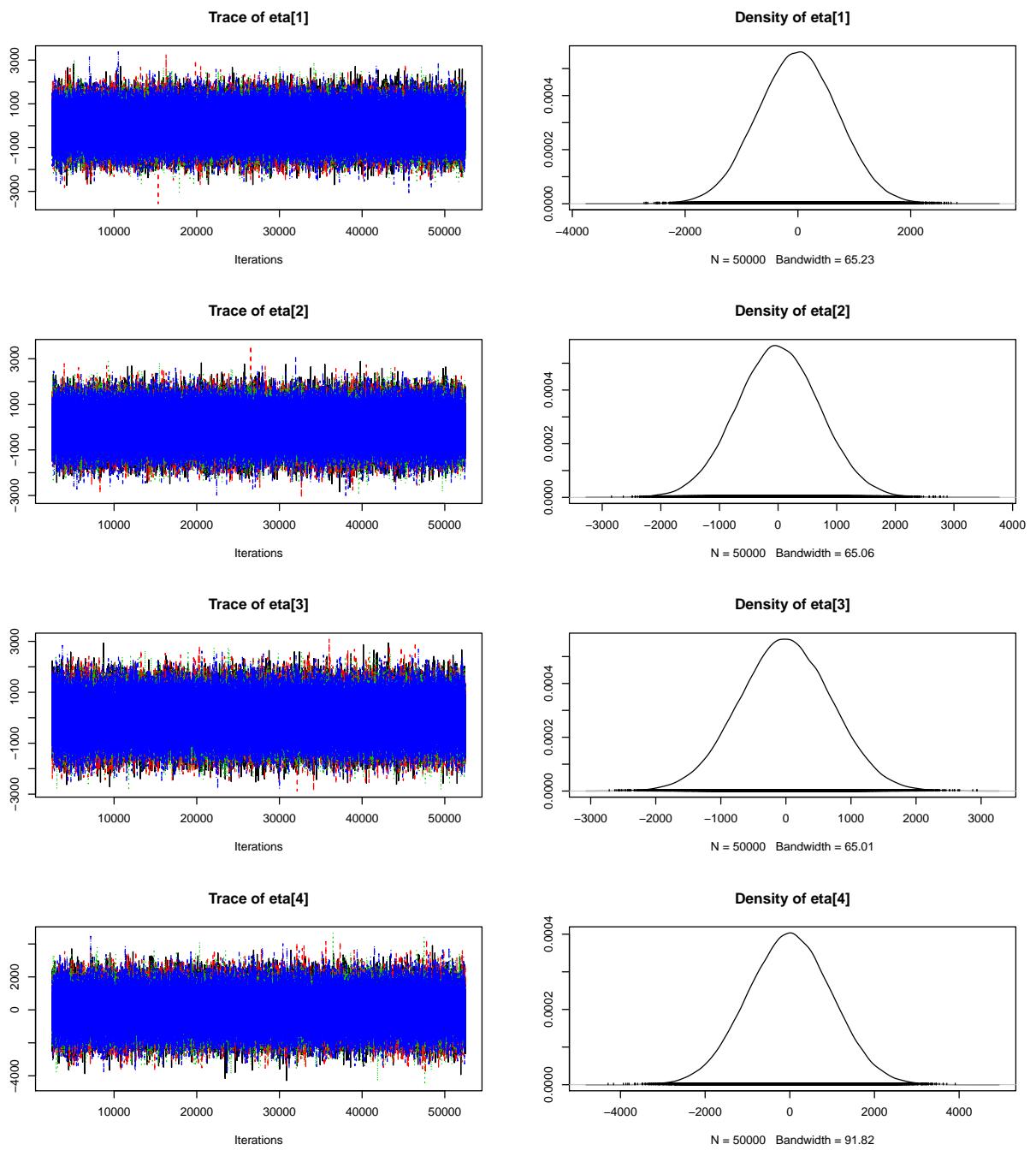


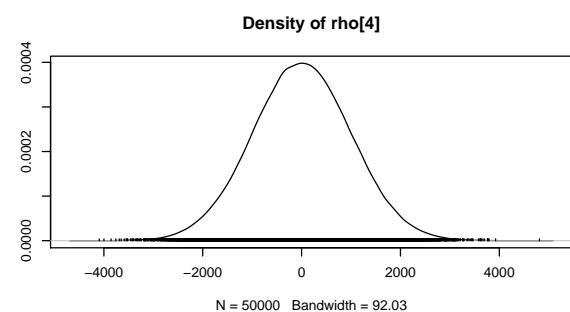
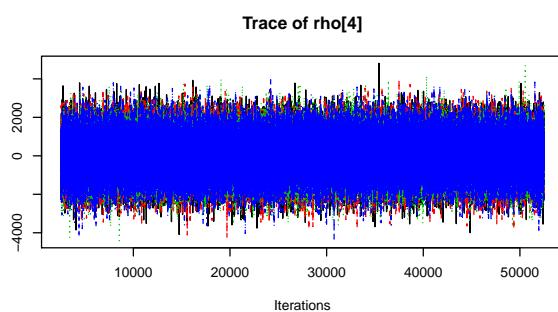
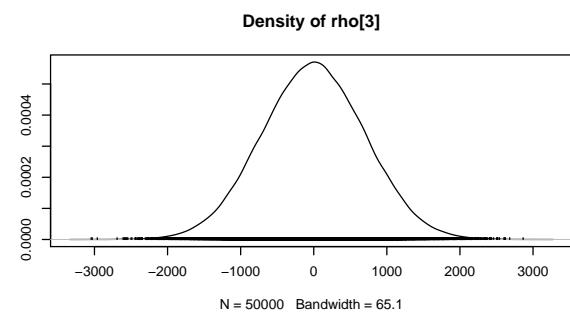
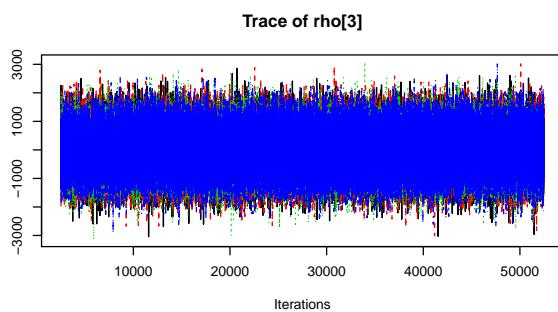
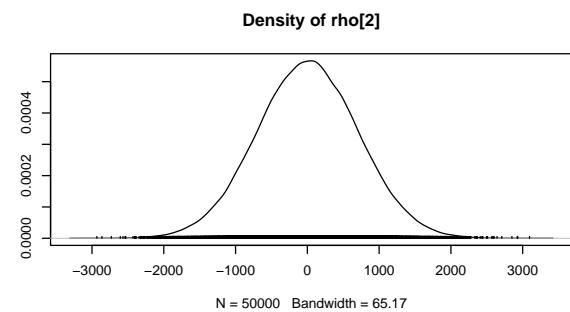
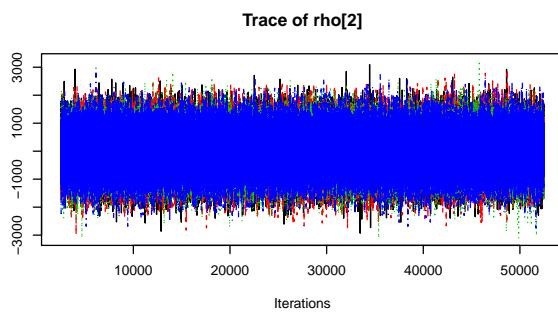
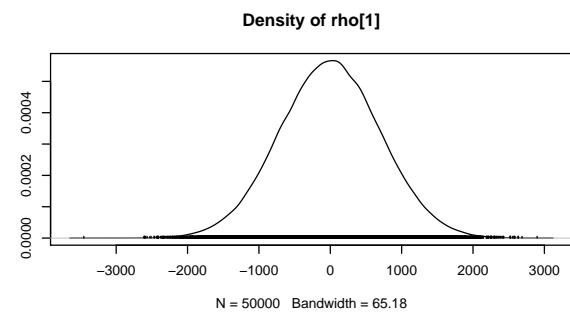
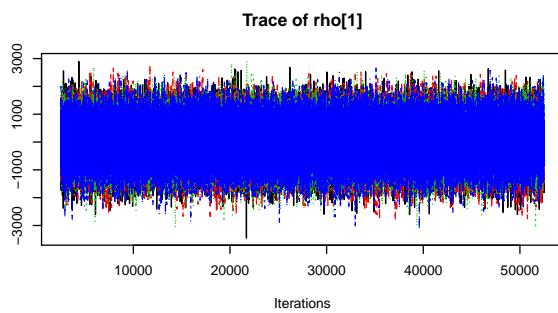


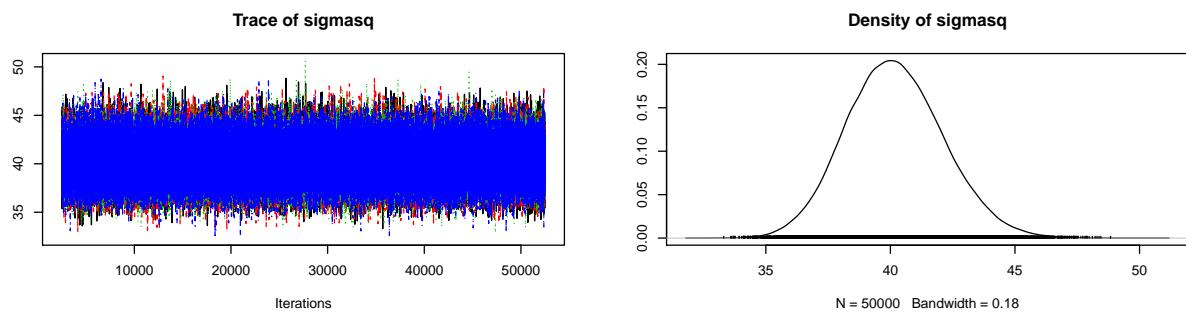












Converged as `gelman.R.max` = 1.0012 < 1.1 and the plot also looks good.

```
summary(result$coda.sam)
```

```
##  
## Iterations = 2501:52500  
## Thinning interval = 1  
## Number of chains = 4  
## Sample size per chain = 50000  
##
```

```

## 1. Empirical mean and standard deviation for each variable,
## plus standard error of the mean:
##
##          Mean        SD Naive SE Time-series SE
## beta.away[1] -4.6182    2.902 0.006489      0.02776
## beta.away[2] -1.2711    1.704 0.003810      0.02644
## beta.away[3]  1.5786    1.730 0.003868      0.02638
## beta.away[4]  3.0885    1.963 0.004389      0.02660
## beta.away[5] -4.4784    1.893 0.004234      0.03548
## beta.away[6] -2.1730    2.367 0.005292      0.03590
## beta.away[7]  1.7616    2.899 0.006483      0.03104
## beta.away[8]  3.6951 1413.491 3.160661      5.45573
## beta.defense[1] -0.0931   0.187 0.000418      0.00309
## beta.defense[2]  0.2060   0.102 0.000228      0.00194
## beta.home[1]   -3.2484   2.648 0.005922      0.02404
## beta.home[2]   -0.5760   1.692 0.003783      0.02608
## beta.home[3]   1.4566    1.718 0.003842      0.02663
## beta.home[4]   3.5874    1.936 0.004328      0.02670
## beta.home[5]   -3.0314   1.851 0.004138      0.03446
## beta.home[6]   -1.5175   2.203 0.004926      0.03305
## beta.home[7]   0.3748    4.114 0.009199      0.03598
## beta.home[8]   0.5614 1410.366 3.153675      5.45185
## delta[1]       -0.9166  996.524 2.228296      2.22831
## delta[2]       -2.4749  996.442 2.228113      2.22812
## eta[1]        -2.6637  706.868 1.580605      1.56449
## eta[2]         0.6431  705.020 1.576473      1.57105
## eta[3]        -4.7849  704.514 1.575342      1.57108
## eta[4]        -1.0183  995.040 2.224978      3.83518
## rho[1]        -3.2953  706.302 1.579338      1.58714
## rho[2]         4.4176  706.207 1.579126      1.57520
## rho[3]        -3.1785  705.498 1.577541      1.57472
## rho[4]         4.4222  997.353 2.230148      3.85933
## sigmasq       40.1765   1.951 0.004363      0.00450
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%   97.5%
## beta.away[1] -1.03e+01 -6.579 -4.6166 -2.6549  1.054
## beta.away[2] -4.60e+00 -2.424 -1.2644 -0.1226  2.049
## beta.away[3] -1.81e+00  0.411  1.5865  2.7517  4.958
## beta.away[4] -7.51e-01  1.761  3.0851  4.4203  6.926
## beta.away[5] -8.20e+00 -5.752 -4.4619 -3.1980 -0.795
## beta.away[6] -6.82e+00 -3.767 -2.1640 -0.5753  2.435
## beta.away[7] -3.90e+00 -0.201  1.7687  3.7207  7.417
## beta.away[8] -2.77e+03 -944.300  1.6354 955.7598 2771.180
## beta.defense[1] -4.57e-01 -0.219 -0.0944  0.0332  0.273
## beta.defense[2]  8.27e-03  0.137  0.2052  0.2744  0.406
## beta.home[1]   -8.44e+00 -5.029 -3.2410 -1.4638  1.940
## beta.home[2]   -3.90e+00 -1.714 -0.5689  0.5648  2.730
## beta.home[3]   -1.91e+00  0.297  1.4670  2.6191  4.815
## beta.home[4]   -2.13e-01  2.282  3.5916  4.8934  7.382
## beta.home[5]   -6.67e+00 -4.275 -3.0132 -1.7765  0.568
## beta.home[6]   -5.86e+00 -2.996 -1.5090 -0.0313  2.766
## beta.home[7]   -7.69e+00 -2.393  0.3835  3.1568  8.411

```

```

## beta.home[8]      -2.76e+03 -950.524  0.2062 952.7990 2766.663
## delta[1]         -1.95e+03 -674.748  0.0143 670.9795 1953.007
## delta[2]         -1.95e+03 -676.399 -0.7245 668.8606 1957.037
## eta[1]           -1.39e+03 -481.560 -1.4531 475.9612 1380.238
## eta[2]           -1.38e+03 -475.825 -1.6285 475.4805 1389.581
## eta[3]           -1.39e+03 -478.837 -5.8174 471.8514 1370.804
## eta[4]           -1.95e+03 -673.567  0.0658 668.8672 1945.829
## rho[1]           -1.39e+03 -478.134 -2.6647 471.2873 1384.093
## rho[2]           -1.38e+03 -471.991  5.2981 479.9338 1391.562
## rho[3]           -1.38e+03 -479.577 -2.9156 472.4013 1379.198
## rho[4]           -1.95e+03 -666.784  2.3488 675.8438 1963.637
## sigmasq          3.65e+01   38.829 40.1158  41.4569   44.157

```

Effective Sample Size

```
(eff.size = effectiveSize(result$coda.sam[, 1:7]))
```

```

## beta.away[1]  beta.away[2]  beta.away[3]  beta.away[4]  beta.away[5]  beta.away[6]
##        11004       4160       4312       5458       2851       4355
## beta.away[7]
##        8743

```

The effective sample sizes of all parameters are greater than 400.